IN2060, Høst 2026

Kapittel 8: Minnehierarki

Philipp Häfliger <hafliger@ifi.uio.no>

30. oktober 2026

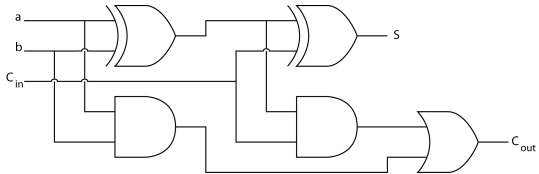
Innhold

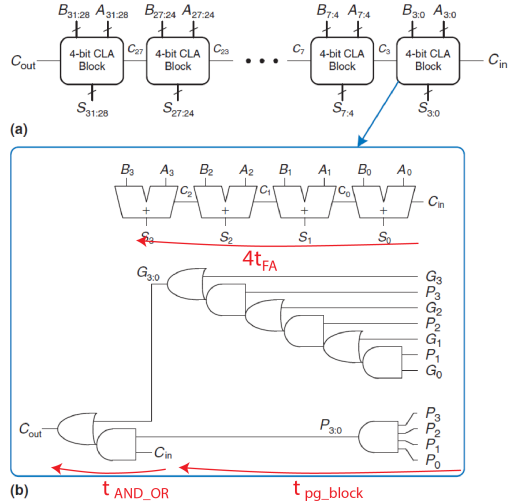
1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Innhold

1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Deloppgave oblig 2: CLA blokk





Lær fra KI, ikke deleger til KI!

Utbredt løsning for oblig 2 som ikke tilsvarer skjematikken som var gitt:

```
for i in 3 downto 0 generate
  p(i) <= a(i) or b(i);
  g(i) <= a(i) and b(i);
end generate;

c(0) <= cin;
c(1) <= g(0) or (p(0) and c(0));
c(2) <= g(1) or (p(1) and c(1));
c(3) <= g(2) or (p(2) and c(2));
c(4) <= g(3) or (p(3) and c(3));

p30 <= p(3) and p(2) and p(1) and p(0);
g30 <= g(3) or (p(3) and g(2))
       or (p(3) and p(2) and g(1))
       or (p(3) and p(2) and p(1) and g(0));
cout <= g30 or (p30 and cin);

for i in 3 downto 0 generate
  fa: fulladder
    port map(
    a    => a(i),
    b    => b(i),
    cin  => c(i),
    s    => s(i),
    cout => open
    );
end generate;
```

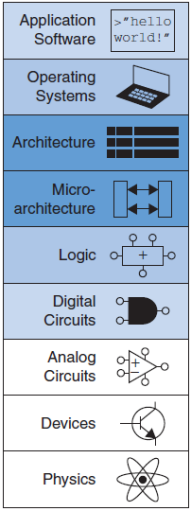
Løsningen forskjellig (hvordan?) men korrekt. Ulemper eller fordeler?

Innhold

1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Kapittel 8 Delmål

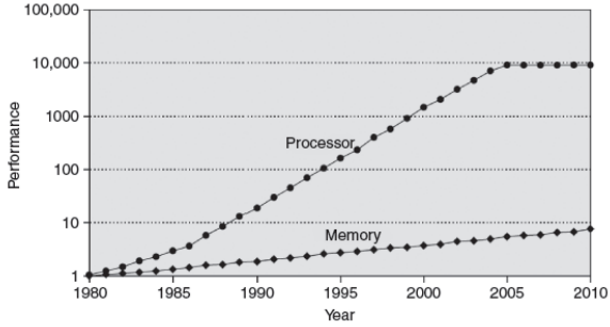
- Forstå hvordan en cache virker prinsipielt
  - Hvordan struktur og størrelse og plassering av en cache påvirker ytelse
  - Hvordan (små) endringer i maskinkode kan gi forskjeller i kjøretid
- Terminologi knyttet til minne og cachesystemer
  - Cache typer og oppbygning
  - Hit, miss, konflikter, ..
- Typisk organisering av virtuelt minne
  - Adresseoversetting
  - Page table
  - Translation lookaside buffer, TLB



Innhold

1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Utvikling av Hastighet, CPU vs Minne



'The memory wall': DDR5 DRAM klarer opp til 70GB/s **MEN** har ca. 16ns forskinkelse, dvs for eksempel en 4GHz prosessor må vente 48 klokkesykkluser på data ved en cache miss, så det koster dyrt.

Minne Trade-Off

Ønskete minneegenskaper er

- rask
- billig
- stor

I praksis får man kun 2 av disse.

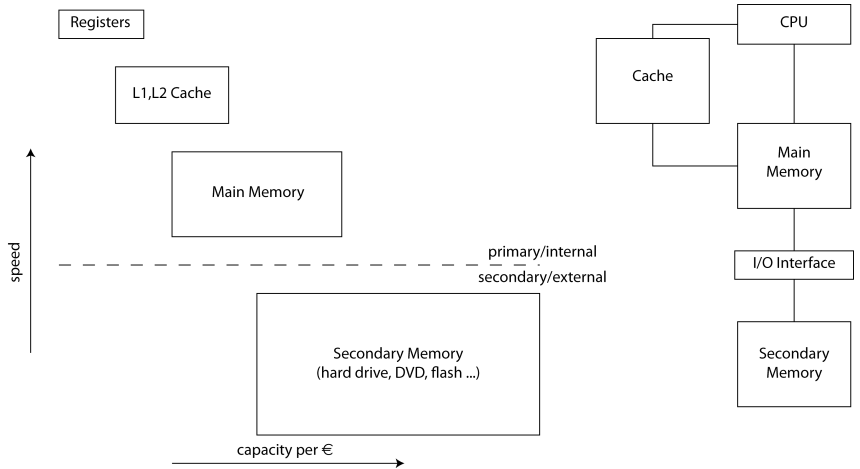
⇒ Minnehierarki



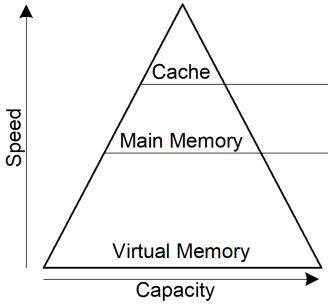
Innhold

1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Minnehierarki



Minnehierarki



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10 - 50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

Mål: data som brukes ofte skal ligge så høyt oppe i hierarkien som mulig.

Minnehierarki skjult for programvare

Minnehierarkiet er vanligvis skjult for programvaren, som ser bare et enhetlig minne-adressområdet. Data på noen av disse adressene er da lagret høyere oppe i cache og det er maskinvaren som velger dynamisk hvilke adresser det er for tiden.
Når kode eller data brukes i minnet skjekkes det om del ligger på øverste cache-nivå. Om det fins der (*cache hit/treff*) brukes det der, om ikke (*cache miss/bom*) skjekker maskinvaren ett minnehierarki-nivå under, osv. Å måtte gå lengre ned i hierarkiet medfører betydelig tidsstraff.

Strategi: utnytte lokalitet

- **Nærhet i tid:**
  Sansynlig at data/kode brukes igjen. Hent og behold nylig brukt data høyt oppe i hierarkiet.
- **Nærhet i adressområdet:**
  Sansynlig at data/kode nær hverandre brukes sammen. Hent større blokker a) rundt måldata som en instruksjon spør om/ b) rundt den aktuelle instruksjonen til høyere nivå i hierarkiet.

Ytelse for minne

- **Treff/Hit:**
  Data ble funnet i det aktuellet minnenivået
- **Bom/Miss:**
  Data er ikke på det aktuelle nivået ⇒ må lete i ett nivå under

$$\text{Hit Rate (HR)} = \frac{\#hits}{\#mem\ accesses} = 1 - \text{MR}$$

$$\text{Miss Rate (MR)} = \frac{\#misses}{\#mem\ accesses} = 1 - \text{HR}$$

- **Gjennomsnittlig aksesstid/Average memory access time (AMAT):**

$$\text{AMAT} = t_{cache} + MR_{cache}\left[t_{MM} + MR_{MM}\left(t_{VM}\right)\right]$$

"*MM*": main memory, "*VM*": virtual memory

AMAT eksempel

En datamaskin med to nivåer: cache og MM.

- $t_{cache} = 1\text{cycle}$, $t_{MM} = 100\text{cycles}$
- $HR = 0.625$

$$\begin{aligned} \text{AMAT} &= t_{cache} + MR_{cache}\,(t_{MM}) \\ &= [1 + 0.375(100)]\ \text{cycles} \\ &= 38.5\text{cycles} \end{aligned}$$

Innhold

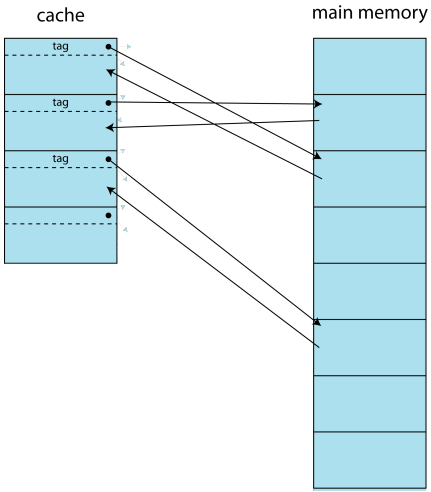
1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Cache

Cache brukes for å forbedre von Neumann minne aksess flaskehalsen. 'Cache' referere til en liten høyhastighets RAM integrert i CPUen eller nær CPUen. Aksesstid til cache er betydelig raskere enn til hovedminnet. Cache-en er liten for å redusere kostnad, men også siden det er alltid en trade-off mellom minnestørrelse og aksesstid. Moderne arkitekturer har flere trinn med cache (L1, L2, L3, ...).
Ellers enn å redusere *MR* så er det også viktig å verifisere om data faktisk ER i cache **fort**. Dette vil bidra til å redusere t_{cache} i AMAT-ligningen, som ikke alltid kan holdes til bare én klokkesyklus. Det fins 3 **hovedstrategier for å mappe** data fra hovedminne (eller nest nederste cache nivå) til cachen:

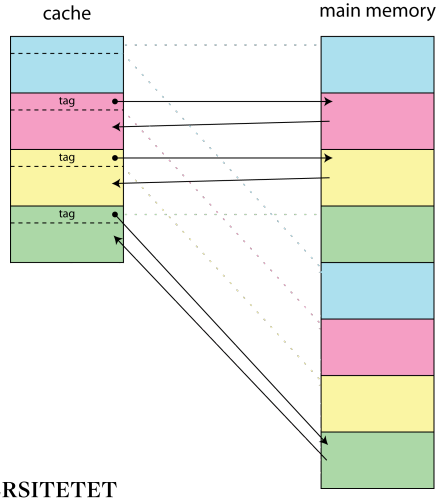
- Associative
- Direct mapped
- Set Associative

Associative Cache



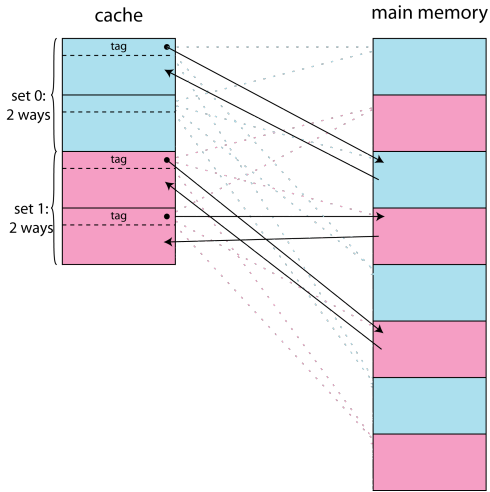
Memory blocks can be anywhere in the cache. The tag is the full block address in the main memory. Conflicts occur only when the cache is full. Determining hit/miss: Parallel search (extremely specialized HW) among memory block tags in the cache.

Direct Mapped Cache



All memory blocks are assigned exactly one slot in the cache by a hash-function (e.g. a number of LSBs of the MM block address (pensum book)). The tag only needs to identify the specific block among all blocks with the same hash-function result (e.g. higher order address bits (pensum book)). Conflicts occur if that cache slot is already occupied. Determining hit/miss: only one tag needs to be checked.

Set Associative Cache

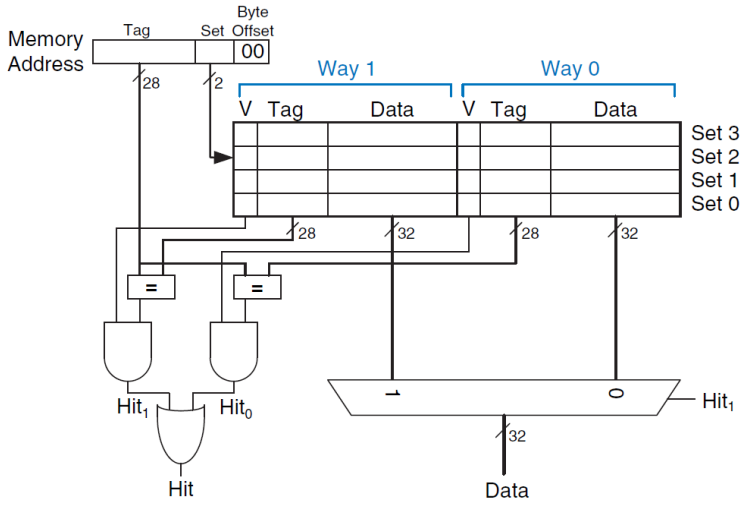


A combination of the previous two: each memory block is hashed to one block-set in the cache. The tag needs to identify the block among all blocks hashed to the same set. Determining hit/miss: Quick search for the tag needs only to be conducted within the set.

Størrelses- og Organiseringsuttrykk

- **Kapasitet** (C, i bytes, kan også oppgis i words)
- **Blokkstørrlse** (b, i bytes, kan også oppgis i words)
- **Antall blokker** ($B := \frac{C}{b} = S \cdot N$)
- **Associativitet** (N, antall blokker per set, si: "N-way set associative")
- **Antall sett** (S)

Set Associative: Utregne en Hit



Koherens og Skrivestrategi

En skriveoperasjon vil medføre en midlertidig inkonsistens mellom innholdet i cache og hovedminne. Det fins forskjellige strategier til å korrigere dette med forskjellige forsinkelser. Hovedstrategiene er:

Write Through: en enkel løsning hvor enhver skriveoperasjon til cache er umiddelbart etterfulgt av en skriveoperasjon til hovedminnet. Som konsekvens vil ikke skriveoperasjoner virkelig ha noen nytte av cache-en.

Write Back: Forsinker skriving til hovedminnen og blokken som er skrevet til i cache-en blir først bare markert som skitten (*dirty*). Bare når en skitten blokk blir gjenbrukt av en annen hovedminneblokk og kastet ut (*evicted*) er den skrevet tilbakke til hovedminnen.

Erstattningsstrategi (1/2)

Som konsekvens av en cache miss en ny blokk behøves å laste inni cache-en. I associative og set associative cache det kan hende at at alle tilgjengelige slots er opptatt og et valg må tas hvilken blokk som skal erstattes i cache-en. (Merk at det ikke er noe valg i direct mapping.) Det fins forskjellige strategier for å ta dette valget, mest prominent er:

- **first in first out (FIFO)**
- **least recently used (LRU)**
- **random**
- **hybride løsninger**

Erstattningsstrategi (2/2)

LRU virker intuitivt og fornuftig men krever en god del administrativt prosessering. Vanligvis et 'brukt' ('used') flagg er satt per blokk når den aksesseres. Med faste intervaller blir den fjernet og en mulighet er da å vedlikeholde en kø med 'ways'-nummerene hvor de brukte blir flyttet bakerst.

FIFO er lettere. Cache-blokkene er organisert i en kø (ring buffer).

random Både LRU og FIFO er i trøbbel om et program går flere ganger sekvensielt gjennom et minnesegment som er større en et set i cache-en: blokken som kastes ut vil snart bli brukt igjen. I sånne tillfeller vil et tilfeldig valg faktisk være bedre.

hybride løsninger, for eksempel ha flere grupper innenfor et set som velges tilfeldig og en FIFO innenfor denne gruppen, osv...

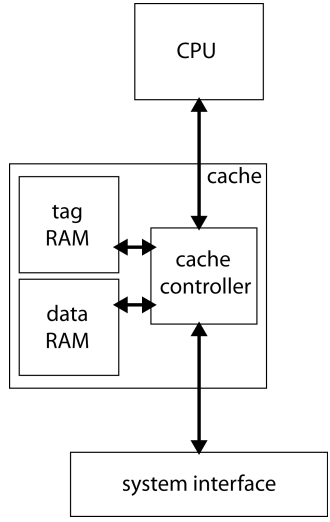
Arkitektur

- **Look Through**
- **Look Aside**

Look Through

The cache is physically placed between CPU and memory

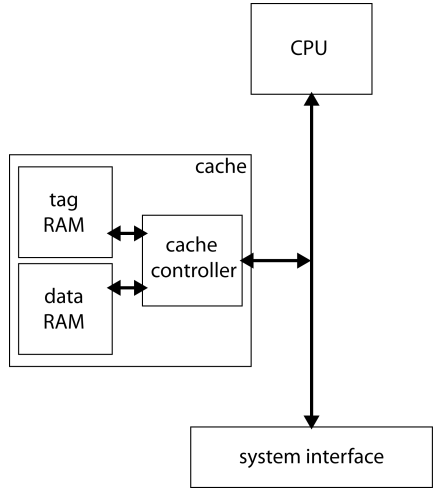
- memory access is initiated *after* a cache miss is determined (i.e. with a delay)
- only if a cache miss is determined, is a memory access initiated
- CPU can use cache while memory is in use by other units



Look Aside

The cache shares the bus between CPU and memory (system interface)

- memory access is initiated *before* a cache miss is determined (i.e. with no delay)
- with a miss the cache just listens in 'snarfs' the data
- only if a cache hit is determined, does the cache take over
- CPU cannot use cache while other units access memory



Innhold

1. Feedback oblig 2
2. Kapittel 8 Delmål
3. Motivasjon
4. Minnehierarki
5. Cache
6. Virtuelt minne vs minnehierarki over

Virtuelt minne vs minnehierarki ovenfor (1/2)

Samspill mellom hovedminne og langtidslagring (som vanlig SSD eller HD) i minne hierarkiet er noe forskjellig fra det mellom cache-nivåene og hovedminne:

1. Cache holder en kopi av en del av innholdet i underligggende minnenivå. Derimot danner hovedminnen og langtidslagring et utvidet **virtuelt minne** hvor innholdet er *enten* i hovedminnen *eller* i langtidslagringen (såvidt det blir diskutert i denne forelesningen). Men vær obs på at pensum boka hinter til noe annet når den diskuterer utvekslingsstrategien, nemlig at det brukes en 'dirty' bit og at en page ikke behøver å skrives tilbake til SSD/HD om den ikke er 'dirty'. Det forutsetter at det er tatt vare på informasjonen om HVOR på SSD/HD denne pagen ligger. $\Rightarrow$ Merk at for dette også operativsystemet (OS) er innblandet i håndtering av en page failure, og det er ikke tema i *denne* forelesningen.

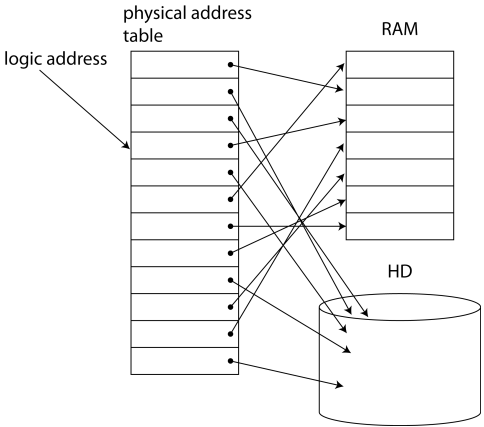
Virtuelt minne vs minnehierarki ovenfor (2/2)

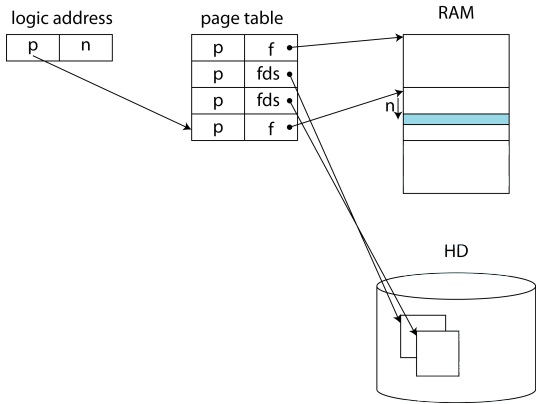
2. En chache miss avgjøres med å samenligne en tag definert av adressen med tags i cache-en. For det virtuelle minnet vedlikeholdes det en **page table** som mapper **virtuelle/logiske minneadresser** til **fysisk minneadresser** (som peker til hovedminnen) eller en file descriptor (som peker til SSD/HD til et området innenfor det som man kaller for bf swap space). **Pages** er typisk 4KB store eller større. Om de er for små ville page tabellen bruke for mye plass. I denne tabellen ser man om page-en er lagret i hovedminnet eller i langtidslagringen. Det sistnevnte heter **page failure** og vil utløse **swapping**, dvs at page-en flyttes fra swap space til hovedminne (og muligens en page fra hovedminne må da flyttes til swap space for å skaffe plass).

Virtuelt minne og minnebeskytelse

3. Virtuel minne management er delt mellom HW og operativsystemet (OS). Det bidrar til **memory protection** om operatrivsystemet tilordner et nummer av MSBs i VM til individuelle prosesser, som bare kan bruke ett mindre adressrom uten disse MSBene. Da har ingen prossess tilgang til minneområdet til en annen.

VM prinsip og ’paging’





Selveste ’paging’-en er håndtert av en **Memory Management Unit (MMU)**

Translation Look-Aside Buffer (TLB)

Siden ’page table’ er ganske stor så holdes det vanligvis i sleveste hovedminnen. Siden hver aksess til VM ville da kreve to aksesser til hovedminnen (page table og selveste data) ville det minst doble adgangstid. Derfor har page table i MMU vanligvis sin egen cache hierarki, og cachen kalles **translation look-aside buffer (TLB)**.

MMU Flow Chart and Block Diagram

